Simple Notes: Access Modifiers & Encapsulation

Access Modifiers

Access modifiers control who can access your class members (variables and methods).

Types:

1. **public** - Anyone can access
2. **private** - Only the same class can access
3. **protected** - Same package + subclasses can access
4. **default** - Only same package can access (no keyword needed)

Quick Reference Table:

Modifier	Same Class	Same Package	Subclass	Everywhere
private	✓	✗	✗	✗
default	✓	✓	✗	✗
protected	✓	✓	✓	✗
public	✓	✓	✓	✓

Simple Example:

```java
public class Student {
    public String name;       // Anyone can access
    private int age;          // Only Student class can access
    protected String school;  // Same package + subclasses
    String city;              // Same package only (default)

    public void showInfo() {  // Anyone can call this method
        System.out.println("Name: " + name);
    }

    private void secretMethod() { // Only this class can call
        System.out.println("This is private");
    }
}
```

Encapsulation

Encapsulation means hiding internal details and providing controlled access to data.

Key Rules:

1. Make variables **private**
2. Provide **public getter/setter methods**
3. Add **validation** in setter methods

Why Encapsulation?

- **Security**: Protect data from unauthorized access
- **Control**: Validate data before storing
- **Flexibility**: Change internal implementation without affecting other code

Simple Example:

java

```java
public class BankAccount {
    private double balance;        // Private variable
    private String accountNumber;

    // Getter method - to read private data
    public double getBalance() {
        return balance;
    }

    // Setter method - to modify private data with validation
    public void setBalance(double balance) {
        if (balance >= 0) {           // Validation
            this.balance = balance;
        } else {
            System.out.println("Balance cannot be negative");
        }
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public void setAccountNumber(String accountNumber) {
        if (accountNumber != null && accountNumber.length() == 10) {
            this.accountNumber = accountNumber;
        } else {
            System.out.println("Invalid account number");
        }
    }

    // Business methods
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Invalid withdrawal amount");
        }
    }
}
```

Using the Encapsulated Class:

java

```java
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();

        // Cannot access directly (compilation error):
        // account.balance = 1000;  // Error: balance is private

        // Must use setter methods:
        account.setBalance(1000);         // Valid
        account.setAccountNumber("1234567890");

        // Use getter methods to read:
        System.out.println("Balance: " + account.getBalance());

        // Use business methods:
        account.deposit(500);
        account.withdraw(200);

        System.out.println("Final Balance: " + account.getBalance());
    }
}
```

Getter and Setter Methods

Naming Convention:

- **Getter**: `get` + VariableName (e.g., `getName()`)
- **Setter**: `set` + VariableName (e.g., `setName()`)
- **Boolean getter**: `is` + VariableName (e.g., `isActive()`)

Simple Pattern:

java

```java
public class Person {
    private String name;
    private int age;
    private boolean married;

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public boolean isMarried() {    // boolean getter uses 'is'
        return married;
    }

    // Setter methods with validation
    public void setName(String name) {
        if (name != null && !name.isEmpty()) {
            this.name = name;
        }
    }

    public void setAge(int age) {
        if (age > 0 && age < 120) {
            this.age = age;
        }
    }

    public void setMarried(boolean married) {
        this.married = married;
    }
}
```

Key Points to Remember:

1. **private** = only same class
2. **public** = accessible everywhere
3. Always make variables **private**
4. Use **public getter/setter** methods for access
5. Add **validation** in setter methods
6. **Encapsulation** = private variables + public methods

7. **Benefits**: Security, Control, Flexibility

Complete Simple Example:

```java
public class Employee {
    // Private variables (Encapsulation)
    private String name;
    private double salary;
    private int id;

    // Constructor
    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    // Public getter methods
    public String getName() { return name; }
    public double getSalary() { return salary; }
    public int getId() { return id; }

    // Public setter methods with validation
    public void setName(String name) {
        if (name != null && name.length() > 0) {
            this.name = name;
        }
    }

    public void setSalary(double salary) {
        if (salary > 0) {
            this.salary = salary;
        } else {
            System.out.println("Salary must be positive");
        }
    }

    // Display method
    public void displayInfo() {
        System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + salary);
    }
}
```

This is the foundation of object-oriented programming in Java!